AMENDMENTS TO THE TITLE:

Please amend the title of the invention as follows:

COMPUTER MEMORY ALLOCATION USING LOOKUP TABLES RESPECTIVELY ASSOCIATED WITH DIFFERENT LEVELS OF MEMORY SEGMENT SIZE

AMENDMENTS TO THE SPECIFICATION:

Page 1, 2nd-3rd paragraphs:

Memory allocators are used by operation systems to allocate free memory upon request from an application. In hardware using a Page Memory Management Unit (PMMU), the memory is divided into fixed-sized memory pages. Accordingly, a simple way of allocating memory is the allocation of free fixed-sized memory pages. One drawback of this approach is that it is inflexible, especially in applications requiring the allocation of small and large memory blocks. As a consequence, memory is wasted.

In general, there are different approaches to deal with the problem of memory allocation. One approach is called "First Fit". It is fast but wastes memory. An example is to allocate memory segments the size of which is the upper power of two values nearest to the requested size. For example, satisfying a request for 2049 bytes results in the allocation of 4096 bytes. Thus, 2047 bytes are wasted. This causes memory fragmentation. After hundreds or thousands of memory allocations and releases, free segments are scattered across the memory and it becomes more difficult to allocate large enough segments, because free segments are small and disjointed, although there is sufficient memory left.

Page 5, last full paragraph:

According to an embodiment of the present invention, an algorithm is provided which is ~~characterised~~ characterized by allocating memory segments from different levels according to their size, using a different granule size (power of two) for each level, and using a multiple-stage bitmap in order to increase the speed when dealing with requests for the allocation of memory blocks.

Pages 5-6, bridging paragraph:

More particularly, memory segments are allocated from seven levels according to their size. [[An]] A range of acceptable segment sizes is defined for each level. In particular, a granule size is defined for each level, and 255 fixed different segment sizes are defined as multiples of the granule size. The largest supported segment size for a given level is

$$maxSegSize=2^N=256\times G$$

where *G* is the granule size of the level.

Page 5, 3rd-10th paragraphs:

Figure 2 is a three-stage bitmap for indicating the state of memory segments;

Figure 3 illustrates a deterministic "Best Fit" memory segment allocation algorithm;

Figure 4 shows an algorithm to determine bitmap indexes;

Figure 5 represents an algorithm to find set bits of the bitmap indicative of a free segment;

Figure 6 is a data structure used in the "Best Fit" memory allocation algorithm;

Figure 7 illustrates a deterministic "Best Fit" memory segment release algorithm;

Figure 8 represents a first doubly linked list linking memory segments; and

Figure 9 illustrates first and second doubly linked lists linking memory segments and free memory segments of the same size, respectively.

Page 7, 2nd-3rd paragraphs:

As indicated above, each level is associated with a table of up to 255 pointers. However, instead of scanning the table of pointers of a level in order to allocate a free [[a]] memory segment, a three stage bitmap is used. Thereby, a deterministic ~~behaviour~~ behavior of the memory allocation algorithm is provided. This approach also considerably speeds up the identification of a free memory segment of the right size.

An exemplary three stage bitmap for use in an embodiment of the present invention is illustrated in Figure 2. The bitmap comprises a root bitmap 1 (first stage), a second stage bitmap 2 and a third stage bitmap 3. The root bitmap 1 is an 8-bit word of which each bit controls an associated 8-bit word of the second stage bitmap 2. If one or more bits of an associated 8-bit word of the second stage bitmap 2 is set to 1, the

corresponding root bitmap bit is also set to 1. ~~Similarily~~ Similarly, each bit of the second stage bitmap 2 is associated with 32 bits of the third stage bitmap 3. If one or more bits of a 32-bit word of the third stage bitmap 3 is set, the corresponding bit of the associated second stage bitmap 2 is also set to 1. Consequently, each bit of the root bitmap 1 represents 256 bits of the third stage bitmap 3. Accordingly, the third stage bitmap 3 comprises 256-bit strings each consisting of eight 32-bit words.

Page 10, 3rd paragraph:

Subsequently, the highest set bit of the 32-bit word 10 is determined. This is bit no. 8, corresponding to an index to the 8th entry of the lookup table 12. The [[eight]] 8th entry of the lookup table 12 indicates the level corresponding to the size of the requested memory block, that is level 0 in the present example. In the next step, the content of the bit of the root bitmap 1 associated with level 0 is determined. In the present example, the level 0 bit of the root bitmap 1 is 0, as the whole memory is free and there is only one free segment of 1 Gigabyte, i.e. all bits of the root bitmap 1 except the most significant one are 0. Due to this result, no AND/SHIFT operation is performed to compute indexes to the second and third stage bitmaps.

Pages 10-11, bridging paragraph:

Then, the lowest set bit of the second stage bitmap 2 is determined. This is bit no. 7, i.e. the highest significant bit of the second stage bitmap 2. ~~Similarily~~ Similarly, the lowest set bit of the 32-bit word of the third stage bitmap 3 associated with bit no. 7 of the second stage bitmap 2 is determined. This is the most significant bit of the 32-bit word, i.e. bit no. 31. This bit is associated with a pointer indicating the physical memory address of the free memory segment to be allocated.

Page 11, 2nd full paragraph:

In the second example, the overall memory size is also 1 Gigabyte. All memory is allocated except one segment in level 0 (say <1024 bytes) and one in level 1 (say <32,768 bytes). A request is received for 18,030 bytes.

Page 14, 1st paragraph:

In the first doubly linked list 17, the order of segments accords with their physical memory addresses. The second double link list 18 includes a number of lists each linking free segments of the same size. Each of these lists is ~~organised~~ organized as a LIFO (Last In First Out) list.

Pages 14-15, bridging paragraph:

As indicated above, when a memory segment is freed, it is merged with ~~neighboured~~ neighbored free segments in order to form a larger free segment. The underlying algorithm is illustrated in Figure 7. When a segment is freed, the state of the ~~neighboured~~ neighbored segments is determined. If both ~~neighboured~~ neighbored segments are free, all three segments are merged. If only one of the ~~neighboured~~ neighbored segments is free, then the two free segments are merged. If no ~~neighboured~~ neighbored segment is free, no merge operation is performed. As a consequence, there are never any ~~neighboured~~ neighbored free segments, as these are merged on freeing one of the segments.

Page 15, 1st full paragraph:

The state of ~~neighboured~~ neighbored segments is determined using the first doubly linked list. The structure of the first doubly linked list 17 is illustrated in Figure 8. Each memory segment has a header 25 which includes information on the state of the segment (free, allocated) and the size of the segment, as well as a pointer pointing to the previous segment. In particular, the state of the segment is indicated by the lowest significant bit of the pointer. A pointer to the subsequent segment is not necessary as its address can be determined from the segment size.

Page 16, 2nd full paragraph:

In addition, two tables of 256 bytes are required to perform computations to determine the first set bit starting from the least and from the most significant bit, respectively. Thus, an additional 512 bytes are required. Further, the doubly linked lists 17, 18 consume 8 bytes per allocated segment or 12 bytes per free segment.

Pages 17-18, bridging paragraph:

As indicated in Table 3, there are three instruction paths for allocating a segment:

-SCBM (Scan Current 32-bit word of the third stage BitMap). In this case, a free segment is found in the current 32-[[but]] bit word of the third stage bitmap 3 (compare Figure 5).

-SUBM (Scan Upper level of the BitMap). In this case, there is no free segment in the current 32-bit word of the third stage bitmap 3, and the current second stage bitmap 2 is scanned to find a free segment (see also Figure 5).

-SRBM (Scan Root BitMap). In this case, there is no free segment in the current 32-bit word of the third stage bitmap 3, nor the current second stage bitmap 2. A free segment is found by scanning the root bitmap 1.

Page 17, Table 3:

	Intel i486 33_MHz	**Pentium 300 MHz**	**PowerPC 300 MHz**
Clock accuracy	+/- 838	+/- 3	+/- 60
Allocate			
Alloc Exact Matching	7000	390	240
Alloc SCBM	15,000	865	540
Alloc SUBM	17,000	1,074	554
Alloc SRBM	17,000	1,144	600
Free			
Free, no merge	6,000	307	224
Free, merge 1 ~~neighbour~~ neighbor	10,000	349	420
Free, merge 2 ~~neighbours~~ neighbors	14,000	795	600

Page 18, 1st full paragraph:

There are also three instructions paths for releasing (freeing) a segment:

-Free a segment while both ~~neighboured~~ neighbored segments are allocated. No merge operation is performed. It is therefore the fastest path.

-Free a segment while there is one free ~~neighbour~~ neighbor. One merge operation is performed.

-Free a segment while both ~~neighboured~~ neighbored segments are free. Two merge operations are performed. This is therefore the slowest path.